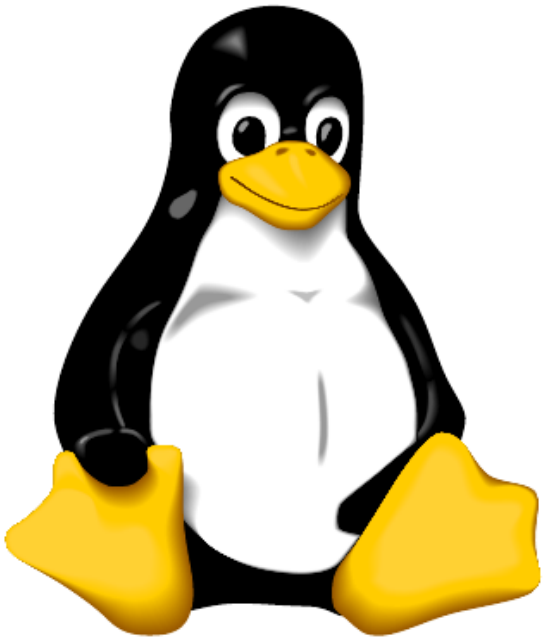
百萬石

MMM'08年春号

人人人人人人人人人人人人人人＿
＞ ゆっくりしていってね！！！ ＜
￣^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y￣



部長挨拶

2008 年度 MMA 部長の村松です. 百萬石をお手に取りいただきありがとうございます.

百萬石春号の発行が新歓期ということもあり, まず, 電気通信大学公認サークル MMA についてご紹介をしようと思います.

MMA, Microcomputer Making Association では, コンピュータに関連する様々な活動を行っています.

新入生の皆さんでも経験があるような, 最新の CPU を発売日に購入してきてパソコンを自作するような行為に始まり, 部の名前の通りに FPGA を駆使して CPU から作るような活動や, 自宅サーバでお手製の Web サービスを運用してみたり, はたまた ACM 国際大学対抗プログラミングコンテストで変なプログラムを提出して怒られるなど, ソフト, ハードを問わず傍若無人の限りを尽くしています.

MMA の部員は主に BSD 系の OS を使用しているようですが, GNU/Linux を信仰する人や Windows をメインに使っている人もいます. これから UNIX を触ってみたいという新入生の方には最適なサークルでしょう.

全ての部員には学内ネットワークへ接続された MMA 共用サーバのアカウントが発行され, 学科計算機室とは違った環境を手に入れることが出来ます. また, 様々な計算機や工具などの揃った部室を使用することが出来ます.

また, 豪華 OB 陣によるアルバイトや就職先の紹介などもありますが, MMA で活動しスキルを身につけた部員は自力で就職してしまうようです.

というわけで, 現在 MMA では部員を募集しています. コンピュータに関心を持つ方ならどなたでも歓迎しますが, 自主的に活動の出来る方を特に求めています.

また, 部室の整理や会計といった部活動を継続する上で必要な仕事を担当できる人も募集中です. 我こそはというかたは是非とも MMA で腕を振るってみてはどうでしょうか.

少しでも興味があれば, mura@mma.club.uec.ac.jp までご連絡頂くか, 4/6 - 4/8 にメインストリートでブースを設けているので, 足をお運び頂ければと思います.

それでは, 私は発売から間もない Phenom 9750 を堪能する作業に戻ります.

目次

MMA鍵システム

oku

oku@mma.club.uec.ac.jp

概要

本稿では、MMAの部室に備えられている**鍵システム** (kagi-sys) について説明する。

kagi-sys はコンピュータ制御された部屋の物理アクセスコントロールシステムであり、要するに、部員なら**いつでも、必要なときに**部室の鍵を開けることができる。鍵としては Suica や PASMO のような非接触 IC カードを採用している。

imo.cltn.org にて開閉状況が確認できる

1 「柔軟な鍵」の必要性

鍵の種類

MMAでは、鍵を**物理鍵**と **FeliCa** で使い分けている。ここで言う、「物理鍵」は通常の鍵、FeliCa は、PASMO や Suica のような IC カードを指す。

物理鍵は、使用頻度の高そうな人に配備し、FeliCa の鍵は原則的にすべての部員が所持している。

そのため、MMAは、部員全員が任意のタイミングで部室を使用可能[*1]というアドバンテージを持っている。

FeliCa 鍵のメリットとデメリット

FeliCa とは SONY による非接触 IC カード技術の名称であり、Suica や PASMO のような交通事業者系カードや Edy のような日本国内で流通している非接触 IC カードの多くが FeliCa をサポートしている。

FeliCa による鍵は以下のようなメリットとデメリットがある。

- メリット
  - 非常に廉価であり、かつ、普及している
  - 無線式であるため、アクセス機器を部屋の内部に設置できる
- デメリット
  - 暗号処理の仕様は公開されていないため、kagi-sys では使用していない
  - 原理的にはカード ID の複製が可能[*2]

ソフトウェアによる鍵制御の有効性

コンピュータによって鍵を制御することは以下の点で従来の物理鍵よりも優れている。

鍵を選択的に廃止できる。例えば、カードそのものを紛失した場合に、紛失したカードだけを無効にすることができる。対して、物理鍵では、すべての鍵を交換しなければならなくなる。

鍵の使用ログを採取できる。カードの ID と実際の使用者を関連付けることによって、誰がいつカードを使用したかを判別し、記録することができる。物理鍵では、誰がカードを使用したかを判別することは出来無い。

鍵の追加が容易である。鍵の追加もソフトウェア的に行う事ができるため、鍵の複製を作成するために店舗まで出向く必要がなく、例えば一定期間のみ有効な鍵などを気軽に発行することができる。

[*1] もちろん、サークル棟そのものが開いている必要はあるが。。

[*2] 通常の暗号アクセスでは問題が無く、あくまで kagi-sys での使用に限った場合

2　システムの実装

モータによる鍵の開閉

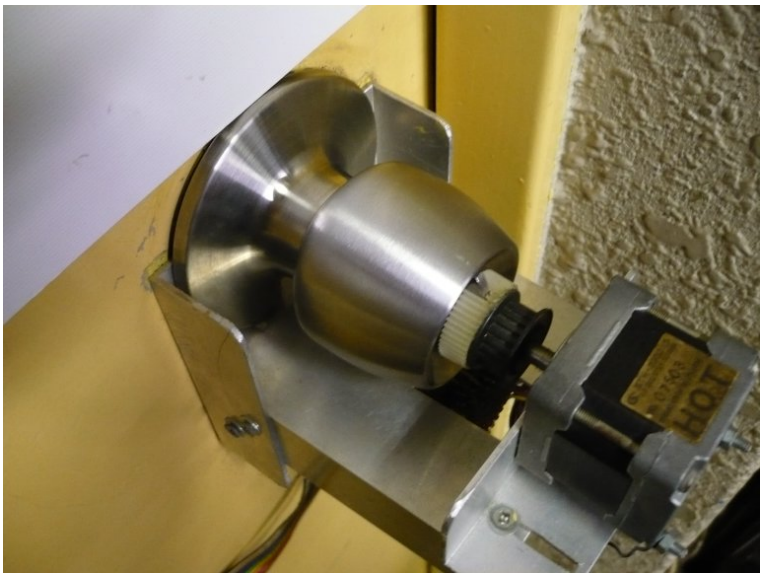


図 1　ドアノブに設置されたモータ

kagi-sys の心臓部は、サムターン部[*3]に取り付けられたステッピングモーターであり、これを利用してソフトウェアからドアの鍵を開閉可能としている。

ステッピングモーターは通常のモータと異り、電気的なパルスを ON/OFF することによって回転させる。そのため、鍵の開閉に必要なパルスの数をあらかじめ調べておくことによって鍵の開閉に必要な角度だけ確実に回転させることができる。

FT245RL によるステッピングモーター制御

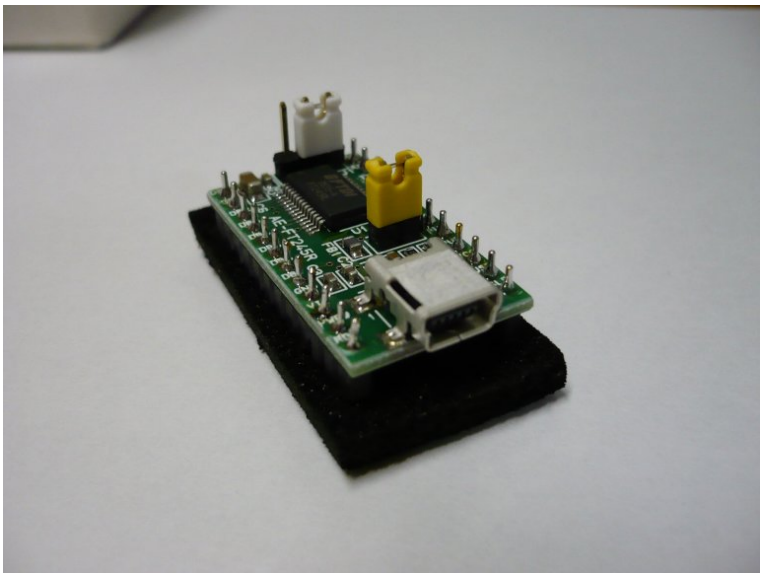


図 2　秋月 FT245RL モジュール

ステッピングモータへ動作パルスを与えるためには、コンピュータから制御可能な TTL レベルの I/O が必要になる。つまり、コンピュータから ON/OFF できる 5V の電圧源が必要となる。この目的のために、FT245RL を使用している。

[*3] 部屋の内側から手で鍵を開閉するためのノブ

従来の kagi-sys では、IP-MOTOR という Ethernet 経由でステッピングモータを制御できる製品を解析の上使用していたが、去年の末破損したため、現在のシステムに置き換えた。

kagi-sys では秋月電子による FT245RL モジュールを使用している。元々FT245RL は表面実装パッケージで工作が面倒だが、この FT245RL モジュールは USB ケーブルで接続するだけで使用できるため、非常に便利である。

FT245RL の制御は FreeBSD の ports にある libftdi を使用している。実際の制御プログラムは数分で完成してしまうほど、システムは単純に記述される。

FT245RL を使用しなくても、同様の事が PC のパラレルポートで行える。しかし、USB は延長や取り回しの面で有利であり、また、近年の PC にはパラレルポートが装備されていない事もある。また、秋月の FT245RL モジュールは 1000 円程度であり、一般的な USB パラレルポートよりも安価と言える。

PaSoRi による FeliCa の読み取り

図 3　SONY PaSoRi

PaSoRi とは SONY による USB 接続のカードリーダーであり、FeliCa の読み取りを USB 経由で行うことが可能となっている。

しかし、PaSoRi はクローズドなデバイスドライバのみ提供されており、さらに、開発に必要な SDK は法人でなければ入手できない等の問題があった。そこで、Windows から PaSoRi を制御させて USB 通信のリバースエンジニアリングを行い、UNIX でも利用可能なライブラリを作成した (libpasori)。

libpasori は、初の FeliCa 関連オープンソースソフトウェアであるとともに、現在存在するいくつかの PaSoRi 制御ライブラリ (libpafe、felicalib) 等の元となっている。libpasori に関しては、ホームページ (libpasori.sourceforge.jp) とオープンソースマガジン 7 月号の特集 2 を参照されたい[*4]。

現在の kagi-sys では FreeBSD の ports に収録されている libpasori を使用している。これは筆者によるオリジナルの libpasori に比べて、共有ライブラリ化や接続された PaSoRi の自動検出などかなりの改善が施されている。

PaSoRi は廊下から直接触れられないように窓ガラス越しに設置されており、このことによってシステムへの攻撃を防いでいる。

kagi-sys では、全ての FeliCa が持っている IDm とよばれる、Ethernet の MAC アドレスに相当するデータを使用している。このため、物理層を直接実装することで詐称が可能であるが、ログは随時メールによって通知されているため、大きな問題は無いと考えている。

[*4] 電通大図書館にバックナンバーがある

3　セキュリティに関する考察

この様なシステムを運用するにあたっては、物理鍵と比較した安全性の検討が必要となる。

kagi-sys は、その動作ホスト (kagi-sys のソフトウェアを動作させている PC) が乗っ取られた場合にはセキュリティを失う。例えば OS のネットワークプロトコルスタックの脆弱性に注意する必要がある。

kagi-sys における最も重大な問題は、認証に用いている ID が一切の暗号的な安全性を持っていないという点にある。また、実際に ID でランダムとなっている部分は 16bit に過ぎないため、総当りで検索することも可能なサイズでしかない。総当たりに対しては一定のカウンターメジャー[*5]を導入して防ぎ、また、鍵の意図しない使用に対しては、鍵を使用したタイミングでメールを送信するという対策を取っている。

実際に、鍵となっているカードのコピーを作成するためには、FeliCa の物理プロトコルを実装し、かつ、カードの ID を何らかの方法で読み取る必要があるが、カードの ID を読み取れるほどカードに接近できるなら、カードの ID を読み取る方法よりも、物理鍵を直接盗む方法の方が有力と言える。

以上から、現在のところは FeliCa による鍵は物理鍵と同等の (少くとも物理鍵を多数複製するよりも高い) 安全性を実現していると考えている。もちろん、今後も FeliCa の運用実態を注視し、定期的に必要なセキュリティが実現されているかを考える必要はある。

現在のところ、不正な入室やアタックは観察されていない。

4　反響

kagi-sys を Wii リモコンでも開閉できるように拡張したバージョンの動画[*6]は engadget や GIZMODO のような海外の有名 blog に取り上げられて非常に多くのアクセスを記録した。

また、kagi-sys のライブラリとして開発された libpasori は、現在でも実世界 UI やビジュアライズ方面で利用されるなどそれなりに利用されており、オープンソースソフトウェアでの非接触 IC カードの利用という面では大きな勢力となっている。

5　まとめ

[redacted]鍵システムは、全ての部員に気軽に部室を利用できる環境を提供し、実際、日常的に活用されている。

おそらく kagi-sys は学内で最もオープンかつ低コストな入退室管理システムであり、この点において[redacted]はかなり先進的な環境を提供していると言える。

[*5] 認証失敗に対して強制的なウエイト等のペナルティを課す

[*6] http://jp.youtube.com/watch?v=ksi4jm_Gkjk

LaTeX 再入門

moechar

moechar@mma.club.uec.ac.jp

はじめに

本稿で扱う LaTeX は厳密には 1993 年にリリースされた LaTeX 2ε を日本語化された pLaTeX 2ε が核となっています。(以後，LaTeX と略記します) レポートや論文作成にあたって初学者にとって敷居が高そうな LaTeX の概念をルー語を交えつつまことしやかに説明します。読者の皆さんを LaTeX に啓発することを目的としているため各節においては包括的に記述してないのであしからず。

1 LaTeX を学ぶ姿勢

トラブルが起きたら安易に他人に訊かず，書籍や web 上の資料からソリューションを見つける努力をすること。エラーが出ても，エラーメッセージの一部をダブルクォーテーションマークで囲って検索すると見つかることが多いです。

即物的なものよりは，アナログかつ basic な手段を好むこと。LaTeX は面倒なことが好きな人や Knuth 病に代表される完璧主義者向けの組版システムです。

現状の自分の LaTeX マークアップスキルに満足せず，絶えずスキルアップを目指すこと。他人にレビューしてもらったりするとエフェクティブです。

2 LaTeX の文献

奥村晴彦さんの LaTeX 2ε 美文書作成入門 (通称奥村本) [1] は初歩から応用までかなり網羅されています。まずは奥村本を一通り読んで LaTeX の用語に対する理解や概念の整理棚を作っておくことをお奨めします。

表現のバリエーションを増やすのであれば中橋一朗さんの解決!! LaTeX 2ε[2] がいいと思います。奥村本と同様，かなり網羅されています。

あとは Knuth 氏の TeX ブック [3] や Lamport 氏 の本 [4] を薦めたいところですが，現在の LaTeX 環境を使うにあたっては即物的というよりは座右の書としてメタ知識を得るスタンスで読むといいと思います。

つい最近，LaTeX 2ε のマクロの本を出され，web 上でも懇切丁寧に LaTeX 2ε の解説をされている吉永徹美さんが翔泳社から独習 LaTeX 2ε という本を出版されたようです。独学向けなんでしょう。多分。

コラム:WYSIWIG とマークアップ

LaTeX はソースファイルをコンパイルして `xdvi` や `dviout` などの dviware を介してレイアウトを確認する形式を取っているので WYSIWIG な環境ではありません。そのためレイアウトを確認して原稿を書いていくためには原稿の平文の内容と LaTeX 命令を同時に頭に浮かべていく必要があります。原稿を進めるたびに LaTeX 命令でマークアップするので平文での思考過程がその都度サスペンドされます。Word のような WYSIWIG な環境と比べると非常に効率が悪い手法と言えるでしょう。

効率よく原稿を進めるのであれば記述者自身が常に LaTeX でしゃべる訓練を積んで従来の方法をエンハンスする 1-pass のやり方と，始めから平文だけで原稿を書き上げ，後から適切にマークアップする 2-pass のやり方が考えられます。前者は短い原稿を書く場合，後者は大規模なレポートや論文を書く場合に有効です。本稿は前者の方法で記述していますがあしからず。

3 LaTeX のインストール

LaTeX 環境のインストール方法を OS 別に説明します。いずれの OS でも以下の物を揃えることが必要です。

- LaTeX 実行プログラム
- dviware (`xdvi`, `dviout`, `etc`)
- `dvipdfmx`
- ghostscript (`gs`)

3.1 FreeBSD の場合

`ports` を使うと驚くべきことに下記の 4 つのコマンドによってインストールが完了します。

```
# portinstall japanese/teTeX
# portinstall japanese/ghostscript-gnu-jpnfont
# portinstall print/dvipdfmx
```

`ports` の依存関係により `xdvi` は自動的にインストールされます。インストールされる LaTeX ディストリビューションは teTeX に日本語化パッチを当てるようにして作られた pteTeX3 と呼ばれるものです。

3.2 Windows の場合

Windows では角藤亮さんが開発, 更新および配布している W32TeX を使用することになります。インストール方法は配布ページに懇切丁寧に記述されているのでそちらを参考にしてください。[*1]

阿部紀行さんの TeX インストーラ 3 を使うと手軽にインストールできます。[*2]

インストールされる W32TeX なるディストリビューションはアスキー社が開発した pTeX を角藤亮さんが x86 命令セットをもつ 32 ビット CPU の元で動作する Windows 環境に移植したものです。

いずれの方法でもインストール後に Windows や `dviout` の環境設定を忘れないようにしてください。

3.3 Linux の場合

私には経験がありませんが，Linux の各ディストリビューションには `apt`, `yum`, `portage` 等の FreeBSD の `ports` に似たパッケージ管理システムがあるのでインストールは容易なようです。[*3][2]

[*1] `http://www.fsci.fuk.kindai.ac.jp/kakuto/win32-ptex/`

[*2] `http://www.ms.u-tokyo.ac.jp/~abenori/`

[*3] `http://oku.edu.mie-u.ac.jp/~okumura/texwiki/?Linux`

3.4 Mac OS X の場合

私には経験がありませんが，Mac OS X にもパッケージ管理システムや TEX インストーラ 3 のようなものがあるようです。[*4][1][2]

3.5 その他の UNIX 系 OS の場合

ソース tarball からの楽しい野良ビルド祭が期待できます。[*5][1][2]

4 新ドキュメントクラス

奥村晴彦さんが jarticle クラスを改良して配布されているドキュメントクラスです。[*6] 従来の jarticle クラスなどよりも優れた組版結果が得られるのが特徴です。百萬石は新ドキュメントクラスをやや弄った物でタイプセットしています。

4.1 インストール

FreeBSD 上の teTEX の場合は ports から容易にインストールできます。ports ツリーを更新してから，

```
# portinstall japanese/platex-jsclasses
```

で一発です。自分のホームディレクトリ以下にインストールする場合は`~/share/texmf` 以下に予め `tex/platex/js` というディレクトリを作った後に奥村晴彦さんのページや CTAN などから取得した新ドキュメントクラスのアーカイブを解凍して中身を全て `js` ディレクトリに入れます。そして `mktexlsr` コマンドを打てばインストール終了です。`mktexlsr` は一度行うとスタイルファイルやクラスファイルを追加するたびに反映させるために実行する必要があるので注意して下さい。

4.2 使用例

```
\documentclass[a4paper,10pt]{jsarticle}
\pagestyle{plain}
\title{test}
\author{hoge}
\date{\today} % コンパイル当時の日付
\begin{document}
\maketitle
ねっ、簡単でしょ?
\end{document}
```

% を用いると，% がマークアップされている場合を除いて右に書かれた文字列が全てコメントアウトされます。% 自体は \% (→%) と書くか`\verb+%+` (→%) 等で印字できます。

5 開発環境

Winshell や EasyTeX, TeXShop, LabEditor といった統合開発環境ソフトを利用するとマクロを搭載したボタンをクリックすることで命令や環境の意味を深く理解せずとも記述できてしまうため初学者にとってはあまり望ましくないと考えています。市販の LATEX 本にありがちな良くないパターンです。

かといって開発効率を考えると，流石にエディターで編集してコマンドラインでタイプセット，というのもいかがなものかという気がします。

自分は Emacs/Meadow 上の YaTeX[*7]を利用しています。YaTeX の利点は Emacs キーバインドを用いて LATEX 文書を記述できるということと，タイプセットや文献データベースや索引の更新，プレビュー等が Emacs 内のキータイプだけで可能な点にあります。Emacs 原理主義な人にはお奨めします。

6 ページレイアウト

LATEX には文書を配置するためのパラメータが存在します。パラメータをデフォルトのままにして組版すると上下左右に広い余白が生じ，行間隔が空いたレイアウトを出力してしまいます。1 段組の文書の最低限設定しておくべきパラメータと設定例を示します。

oddsidemargin : 奇数ページの左マージン
evensidemargin : 偶数ページの左マージン
topmargin : 上マージン
textwidth : 本文領域の幅
textheight : 本文領域の縦の長さ
baselineskip : 行送りの長さ

```
\setlength{\evensidemargin}{-0.5in}
\setlength{\oddsidemargin}{-0.5in}
\setlength{\topmargin}{-0.8in}
\setlength{\textwidth}{7.35in}
\setlength{\textheight}{10.5in}
\setlength{\columnseprule}{0.2pt}
```

なれてきたら 2 段組の文書にもチャレンジしてみてください。えてしてスマートな組版結果を期待できます。

7 図

graphicx パッケージの `dvipdfm` オプションを使うことで EPS ファイル以外に JPEG, PNG, PDF ファイルを文書に取り込むことが出来ます。ただ EPS ファイルとは異なり，表示領域サイズ情報を指示する `.bb`(bounding box) ファイルを生成する必要があります。UNIX 互換環境では

```
% ebb hoge.jpg
```

で `hoge.jpg` の `.bb` ファイルである `hoge.bb` が生成されます。`hoge.jpg` を取り込む方法は EPS ファイルの場合と同様に`\includegraphics[scale=1,clip]{hoge.jpg}`で OK です。

注意

PDF ファイルを取り込む場合は dviware では表示されず，`dvipdfmx` で PDF にしてからでないと表示されないので注意してください。

graphicx パッケージのオプションを `dviout` にしたり，colortbl パッケージを取り込むようにプリアンブルに書くと `.bb` ファイルを生成した時の画像が取り込まれず，コンパイルの段階でエラーが起きるので注意して下さい。

8 表

縦線を引きすぎないということに注意しましょう。日本人の書く表は網掛け過ぎていて海外では不評らしいです。

[*4] `http://oku.edu.mie-u.ac.jp/~okumura/texwiki/?Mac`
[*5] `http://oku.edu.mie-u.ac.jp/~okumura/texwiki/?Make`
[*6] `http://oku.edu.mie-u.ac.jp/~okumura/jsclasses/`
[*7] `http://www.yatex.org/`

コラム:フロート化の是非

図や表を記述する際に無意識に figure 環境や table 環境内に\includegraphics 命令や tabular 環境を記述 (**フロート化**) していることがあります。

図表番号を付けないのであれば必ずしもフロート化することはありません。フロート化する利点は図表の前後の文章のレイアウトと与えられた [htbp] オプションに応じて最適配置してくれることにあります。

一般に，フロート化すると意図した箇所に図表を配置するのが困難になります。フロート化せずに記述すると意図した箇所に配置するのが容易ですが，LaTeX の整形機能によって多少緩和されたとしても前後の文章に関しては適さない形で配置される場合があります。場合に応じて使い分けることが肝要です。

大規模な表を書く場合は Microsoft Excel を使ってみるのも一考です。Excel2LaTeX というアドインを使うとかなり楽に表の LaTeX ソースを作成できます。[*8]

9 カウンター

LaTeX にはページ数，パート数，節数，式, 図, 表番号，文献番号などのカウンターがあり，また記述者が独自にカウンターを定義することができます。section を使うと節カウンターが，*なしの数式環境では数式番号カウンターがインクリメントされます。\setcounter{**カウンター名**}{**数字**}で明示的にカウンターに値を設定することが出来ます。カウンターを用いるとちょっとプログラミング紛いなことができます。[2]

10 相互参照

相互参照は label 命令でラベルが記述された場所に存在するカウンタ値を ref 命令によってラベルを参照して拾い上げるシステムです。このシステムでは ref 命令を対象とするラベルの前に書くことが出来るので TeX 以前の時代と比べて大変画期的なシステムであるといえます。

10.1 図, 表の場合

figure, table 環境内にラベルを記述し，そのラベルを ref することで図, 表番号が出力されます。

10.2 数式の場合

equation, align といった数式環境内にラベルを記述し，そのラベルを ref することで数式番号が出力されます。eqref 命令を使うと (14) というように括弧付きで数式番号が得られます。語尾に*が付く環境では数式番号がインクリメントされずに表示されないのでラベルを貼って ref すると??となるので注意して下さい。

10.3 節などの場合

section，subsection，subsubsection などの節を構成する命令の後にラベルを記述し，そのラベルを ref することで節番号が出力されます。節の場合も語尾に*が付く命令の後にラベルを貼って ref すると??となるので注意して下さい。

[*8] http://www.ctan.org/tex-archive/support/excel2latex/

10.4 ページ

ページ数を参照する場合，参照したい項目に対してラベルを記述する以外には前節までのようなラベルを新たに記述する必要はありません。ラベルに対して pageref 命令で参照することによってそのラベルが位置するページ番号が出力されます。

実はページを相互参照するケースは非常に稀です。「104 ページの図 5 を見ると必然的だが図 6(105 ページ) を見ると恣意的だ」のようにくどいからだと思われます。

10.5 記述例

```
〜を図\ref{fig:hoge}に示す.
\begin{figure}[htbp]
 \centering
 \includegraphics[scale=1,clip]{./hoge.jpg}
 \caption{test}
 \label{fig:hoge}
\end{figure}

図\ref{fig:hoge}より\eqref{eq:hoge}式がなりたつと推測できる.
\begin{equation}
 I(X;Y) = H(X) + H(Y) - H(X,Y) \label{eq:hoge}
\end{equation}
```

編集中にカウンタの値が代わってしまう場合は原稿の LaTeX ファイルを 2〜3 回 platex で処理する必要があるので注意してください。1 回だけだとカウンタ値が.aux ファイルに書き込まれる前に ref 命令でカウンター値を参照し，??と表示されてしまいます。

11 参考文献の記述方法

参考文献を記述する方法には thebibliography 環境を使う方法と jBibTeX よる方法があります。前者は空でもタイプできる程度の数の文献を載せるときに使い，後者は文献が膨大な数に上る場合に使うと効果的です。本稿では前者の方法を説明します。後者についてはシステマティックに記述されている [1],[2] を参照してください。

11.1 書籍の場合

(1) 文献のラベル (\bibitem{}の中身)
(2) 著者名
(3) 書籍のタイトル
(4) 出版社
(5) 出版年月

を書くことが推奨されています。

11.2 論文などの場合

(1) 文献のラベル (\bibitem{}の中身)
(2) 著者名
(3) 論文のタイトル
(4) 論文を収録している冊子名
(5) 冊子の巻数
(6) 冊子が刊行された号数
(7) 論文が冊子の中に載っているページの範囲
(8) 冊子が刊行された年月

を書くことが推奨されています。

```
\begin{thebibliography}{99}
 \bibitem{bib:oku} 春山 晴彦，入門 SSH，アスキー，2004 年 12 月
 \bibitem{bib:kol} A. Kolmogorov, ‘‘Logical basis for information theory and probability theory",
{\em IEEE Trans. Inform. Theory}, Vol.~14, No.~5, pp. 662--664, Sep 1968.
\end{thebibliography}
```

図 1　参考文献の記述例

11.3　引用方法

参考文献のカウンターを参照するには cite 命令を使います。ref 命令と同様に文献ラベルを引数に与えることで文献番号を参照します。したがって，組版後に編集中に文献の数を増減した場合は相互参照の場合と同様に原稿の LaTeX ファイルを 2〜3 回 platex で処理する必要があります。

LaTeX	\cite{bib:kol}で主張されていることは · · ·
実表示	[2] で主張されていることは · · ·

12　スタイル · クラスファイルの利用方法

本節ではスタイルファイルやクラスファイルの基本的な利用方法を説明します。LaTeX ディストリビューションの取得時に含まれていないスタイルファイルやクラスファイルは CTAN 等で取得できます。

12.1　スタイルファイル

LaTeX ディストリビューションの取得時に含まれいてるスタイルファイル (パッケージとも呼ばれます) や，LaTeX ソースファイルと同じディレクトリにあるスタイルファイルを使用する場合はプリアンブルに

\usepackage[**オプション**]{**スタイルファイル名**}

と書けば利用できます。スタイルファイルとして hoge パッケージ (hoge.sty) を使いたい場合はスタイルファイル名は拡張子を除いた hoge となります。

LaTeX ディストリビューションの取得時に含まれていない LaTeX ソースファイルと異なるディレクトリにあるスタイルファイルを新規に使う場合は 4.1 節で説明した様にスタイルファイルを適切なディレクトリに格納し，mktexlsr を実行した後に上記の方法で利用できます。

原稿を記述するに当たって関係ないスタイルファイルまで取り込みすぎるとそれら同士が干渉してトラブルの元凶になることがあるので注意してください。

12.2　クラスファイル

4.2 節の 1 行目の様に

\documentclass[**オプション**]{**クラスファイル名**}

と書けば利用できます。クラスファイルとして foo クラスファイル (foo.cls) を使いたい場合は拡張子をクラスファイル名は除いた foo となります。LaTeX ディストリビューションや LaTeX ソースファイル云々の話は 12.1 節と同じです。

13　数式

amsmath パッケージを使うことで複数行立てスタイルの数式環境の align 環境が利用できます。align 環境は eqnarray 環境よりもスマートに数式を記述できます。

```
\begin{eqnarray}
 I(X;Y) &=& H(Y) - H(Y|X)\\
        &=& H(X) + H(Y) - H(X,Y)
\end{eqnarray}

\begin{align}
 I(X;Y) &= H(Y) - H(Y|X)\\
        &= H(X) + H(Y) - H(X,Y)
\end{align}
```

両方とも同じ数式ですが，利用するクラスファイルによって組版結果が異なります。jarticle クラスでは = の左右に空白が入りますが，jsarticle クラスでは空白が入りません。

数式や記号表示に関しては LaTeX ネイティブのものよりはアメリカ数学学会が提供する $\mathcal{AMS}$-LaTeX の方が多彩に表現できます。amsmath パッケージと amssymb パッケージを読み込むことで利用できます。

align 環境のほかには alignat や multiline 環境，cases パッケージの numcases 環境などがあります。

14　組版後にやること

dvi ファイルのまま印刷してもいいのですが，文字や画像がジャギー等のギザギザが目立って印刷されてしまいます。dvipdfmx や dvips + ps2pdf などで PDF 形式にするとアンチエイリアスが効き，見栄えがよくなります。

さらに，-f オプションや utf/otf パッケージ等を利用して PDF ファイルにフォントを埋め込めば，可搬性が向上するばかりか表示先のフォント環境依存性による心配もなくなります。

15　おわりに

本稿を読んだ後に [1] や [2]，脚注で挙げた web page を読まれるとよりいっそう理解が深まると思います。逆もまた可か否かは読者の皆様に委ねます。

LaTeX に関する体系的な知識は [1] の他には記事の中で何度か脚注で挙げた奥村晴彦さんの TeX Wiki に集大成されています。表現については物理のかぎしっぽさんの web page[*9]や吉永徹美さんの LaTeX 入門[*10]が大変参考になります。

参考文献

[1] 奥村 晴彦, LaTeX 2_ε 美文書作成入門 改訂第 4 版, 技術評論社, 2007 年 3 月

[2] 中橋 一朗, 解決!! LaTeX 2_ε, 秀和システム, 2005 年 11 月

[3] Donald E. Knuth, [改訂新版] TeX ブック, アスキー, 1992 年

[4] Leslie Lamport, 文書処理システム LaTeX 2_ε, ピアソン · エデュケーション, 1999 年

[*9] http://hooktail.org/computer/index.php?TeX

[*10] http://www.h4.dion.ne.jp/~latexcat/intro/intro-index.html

MMAなんだからコンピュータ作ろう (FPGA で)

oku

oku@mma.club.uec.ac.jp

1　自作ってどういう事か

自作 PC が流行ってるけど。

よのなかには自作 PC という言葉があって、主に、『既存のマザーボードとかを買ってきて、組み立てる PC』の意味で使われています。実際、MMAでも、普段使ってる PC の殆どはメーカー製の PC ではないのです。

しかし、それって『自作』なんでしょうか。

いや、『手作りクッキー』が小麦から育てていないように、適当なところで足切りするのはそれほど間違っていないと思いますが、やっぱり、そういうのって単に『組み立て PC』に過ぎないよなぁとか思うわけです。

今日の PC は様々な部品 (ソフトウェア、ハードウェア問わず) で出来ています。

- CPU
- OS
- HDD
- キーボード
- etc..

注目すべきなのは、『自作』PC はこれらのいづれも作ってない[*1]ということです。どこに創造性があるというのか。部品の選定？

他人と違うところで創造性を発揮しましょう。

2　計画

遊びの範囲でどこまで出来るのか

Intel くらいの体力があれば、俺アーキテクチャに世界を従わせることができます[*2]。しかし、通常の人間には IC を設計したりそれを製造したりするのは不可能でした。

現在の技術はそれを可能にしています。多少の知識があれば、一昔前の CPU くらいの規模の IC なら普通のプログラミングと同じようにソースコードを書いてそれをコンパイルすることで作れちゃいます。つまり、完全たる俺 CPU を作ることは可能です。

むしろ、キーボードを自作する方がずっと難しいでしょう。100 以上のスイッチを半田付けするなんて！

OS は単なるソフトウェアなので、自作するのは簡単です。

はりぼて PC[*3]

とりあえず、何もかも手作りするのではそれだけで人生が終ってしまうので、とりあえず 1ヶ月くらいで作れる範囲のスペックを決めました。

- 32bit デュアルコア RISC-CPU
- USB1.1 ホストコントローラ
- RAM インタフェース
- VGA インタフェース

[*1] 手作りクッキーと同様に

[*2] 実際には IA-64 がどの程度成功しているかを考えればわかるように、Intel といえど新しいアーキテクチャを普及させるのはむずかしい問題です。

[*3] 『OS 自作入門』へのオマージュとしての名称

これだけです。はりぼてもいいとこですが、今 USB で手に入る周辺機器の数を考えてみれば、これだけあれば通常の人間生活が送れることに気付くと思います。

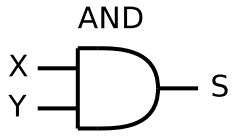
これら 4 つ全て、秋葉原にでも行けば IC として入手できるんですが、まあ、そこを手作りするのが醍醐味ということで。そして、全てのパートを 1 つの FPGA で作ります。

3 論理回路をさわりだけ

そもそも IC は何をやってんのか (Level 0)

コンピュータは 0 と 1 で計算している、と聞いたことはあると思います。こういう 0 と 1 を処理する回路のことを**論理回路**といって、実際、コンピュータはだいたいのところ論理回路でできています。

論理回路を構成する要素としては、AND とか OR とか XOR(排他的論理和) とか NOT とかがあります。極論を言えば、これらを組み合わせてひとつのパッケージに収めたのが IC[*4]です。

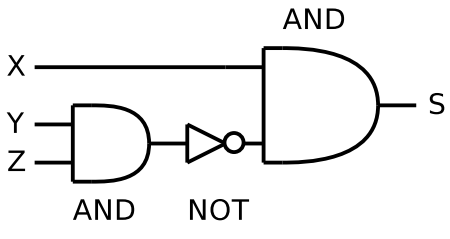


X	Y	S
0	0	0
0	1	0
1	0	0
1	1	0
1	1	1

図 1 AND の MIL 記号と真理値表

例えば、AND は⊐のような記号で書かれます。2 入力の場合の表が図 **1** にありますが、要するに入力が両方 1 にならないと 1 をださない奴であるという意味です。こういう表のことを**真理値表**っていいます。なんか宗教がかった言い方ですが、これは 1 を True(真)、0 を False(偽) と呼ぶ習慣が有るからです。他にも要素は OR とか NOT とか有りますが、要するに真理値表の結果の部分が違うだけです。

どんなに回路の規模が大きくなっても、真理値表を大きくしていくだけ。簡単だね (図 **2**)。



X	Y	Z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

図 2 複雑な回路と真理値表

じゃあわざわざ、AND とか OR とか NOT を秋葉原で買ってきて繋げる[*5]よりも、**真理値表だけで回路を表現したほうが楽だよね-というのが FPGA** です。FPGA をつかえば、数十万もの AND とか OR を無料で組み合わせて使え (るのと同じ状況を作り出せ) ます。

しかし、AND とか OR をどうやって組み合わせれば CPU になるのかというのは、なかなか想像がつかないでしょう。もう一段階上の階層を考える必要があります。

[*4] IC には、もちろん計算以外を目的とした物があります。そういうのは、例えばアナログ回路とかもっと別のものが入ってるでしょう。

[*5] 74 シリーズとか言ってるのがこの類です。IC としてマジで AND とか OR が売ってます。もっと複雑なのもありますが。

組み合わせ回路 (Level 1)

超端折って説明すると、AND とか OR とか NOT を複雑に組み合わせると『2 進数で表現した数を入れるとその合計が 2 進数で出てくるような回路』を作ることが出来ます。例えば、2bit + 2bit の計算をする回路を考えると、入力が 4bit で出力が 1bit の真理値表を 3 つ考えれば良いって事です。

この辺の設計はコンピュータが勝手にやってくれる便利なソフトがあるので、それを使います。

そして、CPU を作るためには、足し算器とか引き算器を並べて、適宜繋ぎかえてしまえば良いって事になります。簡単だね。しかし、単純に並べるだけでは問題が起きます。

同期回路と非同期回路 (Level 2)

さて、光や電気は速いけど伝わるのには時間が掛ります。つまり、足し算器や引き算器を作って繋げたところで、結果が出てくるまでに適当な時間が掛るだけでなく、**足し算と引き算でかかる時間が違うことも考えられる**わけです[*6]。これでは、次の仕事にいつ取り掛っていいのかわかりません。足し算しかできないコンピュータなら別ですが。

これには 2 つの解決策が考えられます。ひとつは『次の仕事をしていいよ信号』を考えて、それを受けとったら次の仕事をするという方式。しかし、これはメジャーでは無いです。もうひとつが一番遅そうな奴に合わせる、つまり、**クロック**を使う方式ということになります。

クロックは、一般に一定周期で 0 と 1 を繰り返す信号で、『1 になったら次の仕事をする』と約束するのが普通です。逆に言えば、足し算器や引き算器はクロックが切り替わるまでの時間で、仕事を終える必要があります。

こういう、何らかの信号を使って、「せーの」で動く回路の事を**同期回路**といいます。そうでないのが**非同期回路**。

よく、CPU のスペックでクロックというのは目にします。僕がこの原稿を書くのに使っている PC の CPU は 400MHz で動作しているので、一つの仕事を 400 メガ分の 1 秒間に終えてると考えられます。もっとも、近代的な CPU は仕事をこまかい単位に分割することでクロックを上げやすくしています (パイプライン化って言います)。つまり、1 秒間に 400 メガ回の足し算が出来ると単純に考えることは出来ません。

これで CPU が作れるかというと、もう一つだけ必要な概念があります。

順序回路 (Level 3)

今まで説明した回路は基本的に全部、『今の入力によって出力が決定する』回路です。しかし、実際の回路を作るに当っては、『過去の入力と今の入力によって出力が決定する』回路が必要になります。要するにメモリの類です。こういうのを**順序回路**っていいます。

通常想像されるのはメモリでしょう。つまり、一度入れたデータを覚えて、出力に出しつづけるという動作をします。一般の人間はこういうのをメモリといいますが、論理回路の世界では、**レジスタ**とか、**フリップフロップ**っていいます[*7]。フリップフロップは 1bit のメモリと考えることができます。1bit しか無いとアドレスを指定する必要は無いですね。

これで、CPU を作るのに十分な材料は揃いました。しかし、通常の人間はそこまで原始的なレベルで考えて物を作るのは面倒なので、もう一段階抽象化を進めます。

高機能な機能ブロック (Level 4)

実際の設計では、もっと高機能なブロックを組み合わせて CPU を作っていく事になります。まあ、面倒だしね。。

たとえば、フリップフロップを並べて、『どのフリップフロップの信号を取り出すか』という信号 (要するにアドレス) を付け加えれば普通のメモリになります。ほかにも、足し算器とかカウンタとか良く知られた回路を組み合わせるのが好ましいです。

[*6] コンピュータに詳しいなら、ここで、「引き算は 2 の補数の足し算でいいじゃないか」と突っ込みを入れたくなるかもしれません。

[*7] これも 74 シリーズにあるので、秋葉原で買ってこれます。原理上は。

4　CPU にする

すでに調理したものが。。

ここで、何を考えて命令セットを作ったとかそういう話を始めるとそれだけで一冊書ける (というか書いてる) ので、ここでは、簡単に、CPU がどういう構造をしているのかだけを書きます。

図 **3** が、構成を表わした図です。

CPU の機能の中心となるのは、ALU という回路で、これは、命令 (スルーパス、足し算、引き算、かけ算、等) を表わす入力と、処理対象の数 2 つを入力すると、そのうち出力から処理結果が出力されるという動作をします。もうこれだけで CPU になってしまいそうですが、まだ仕事が残ってます。

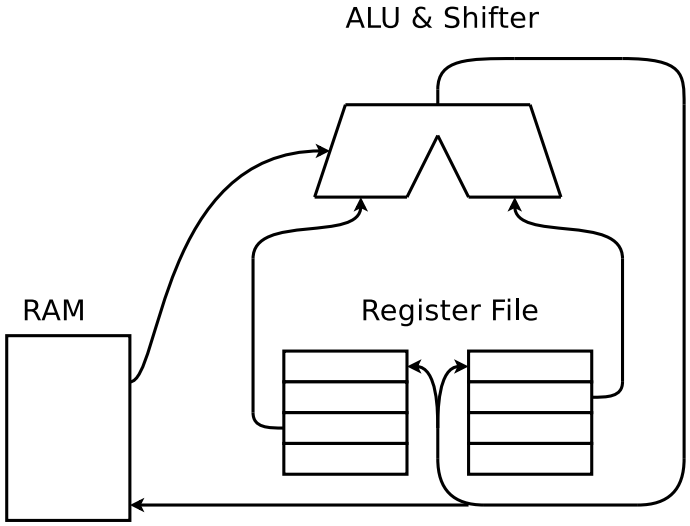


図 3　CPU の構成 (笑

残りの仕事は、『ALU に喰わせるデータを用意する』ことで、このために RAM と Register File にデータを溜められるようにします。

Register File というのは、小さな RAM のことで、ALU に直接値を入力するためには、この Register File にデータをコピーする必要があります。この図ではこのデータの流れを省略していますが、この目的のための専用の回路を別に用意します。

Register File は 2 つありますが、これは同じ内容のものを 2 つ用意することにしています。何故かというと、Register File は 2 つの読み込みと 1 つの書き込みを同時に処理する必要があるのに、FPGA の中の RAM は 1 つの書き込みと 1 つの読み込みを同時に処理することしか出来無いからです。

命令も RAM に入れるので、現在実行している命令のアドレスを保持するレジスタが必要になります (これも図では省略しています)。これは慣例的に**プログラムカウンタ**と呼ばれます。また、同じ処理を繰替えしたり、条件判断による分岐などが出来無いとかなり困るので、プログラムカウンタも操作できるようにしておきます。

CPU の仕事をまとめると、次のようになります。

- (計算モードのとき)
  - ALU に喰わせるデータの入ってるレジスタを指定する (2 つ)
  - ALU の結果データを入れるレジスタを指定する (1 つ)
  - ALU の作業内容を指示する
  - (もし必要なら) 結果を入れる RAM のアドレスを指定する
- (RAM アクセスモードのとき)
  - RAM からのデータを入れるレジスタを指定する
  - RAM のアドレスを指定する
- (ジャンプモードのとき)
  - プログラムカウンタを ALU の結果に設定する

あとは、RAM に↑のような指示 (いわゆる**機械語**) を並べていけば OK です。

かなりあっけないですが、実際、CPU を構成すること自体はそんなに難しくなくて、速度さえ気にしなければちょっと論理回路の授業を受けていくつかのプログラミング言語を習得すればすぐできます。[*8]

5 FPGA での実現

FPGA とは

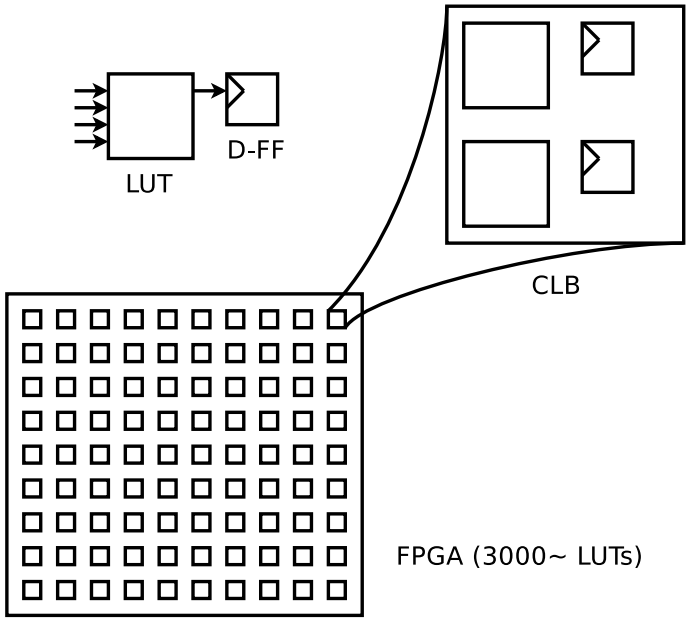


図 4 FPGA のなかみ

FPGA は、RAM をつかって論理回路をかなり自由につくることができる IC で、

- 図 **4** のような LUT と D フリップフロップ
- 普通のメモリ
- 専用の乗算器
- 等々

を自由に接続して使えます。実際の接続は碁盤の目状にあらかじめ配線が行われているので、その配線の交点のうち、どこを使うかを指定することで行います。もちろん、こういう配線はプログラムが自動的にやってくれます。

いろいろ便利な機能ブロックも予めはいってますが、大部分のロジックは LUT とそれにくっついたフリップフロップで作ります。

LUT には真理値表を入れます。僕の普段使っている FPGA は 4 入力 LUT を採用しているので、各 LUT は 16bit($2^4 = 16$) のメモリって事です。LUT は非同期回路なので、同期回路を作るためにフリップフロップが先についています。これにクロックを入力して、まわりの LUT に「せーの」で結果を入力します。

以下、用語は全部 Xilinx という FPGA メーカーの奴にあわせますが、他のメーカーのでも大筋はおなじです。

LUT はいくつかを組み合わせてスライスとか CLB という単位にします。この単位の LUT は繋げて使えたり、他所よりも密接に連携できます。隣同士の CLB も自由に配線できることが多いです。CLB を格子状に並べたのが FPGA ということになります。

[*8] ただ、既存の CPU と同じものを作ろうとするのはかなり難しいので、どうにかして、誰でも簡単に作れてかつ実用的な CPU を作るというのがこの研究の目的の一つでもあります

回路の設計方法

FPGA に載せる回路を作るには次の 4 ステップを踏みます。が、人間がやるべきなのは最初の一つだけです。

(1) デザイン入力
(2) 論理合成
(3) マップ
(4) 配置と配線

デザイン入力というのは、回路図を直接入力するって方法もあるんですが、普通は VHDL か VerilogHDL という**ハードウェア記述言語** (HDL) を使って回路の『仕様』を記述します。

元々、HDL というのは「作った回路の動作を記述する」言語、もしくは、シミュレーション用の仕様記述言語だったので、回路そのものを記述する言語ではなかったのです。**論理合成**はその元々の目的から見れば逆の行動、「動作の記述から回路を作る」という作業です。これができるおかげで、いちいち AND とか OR の式をずらずらと書かなくても、コンピュータが自動的に AND とか OR で構成された回路に変換してくれます。

マップは、合成された回路を LUT に入れる真理値表に分割する作業です。これもコンピュータが勝手にやります。 配置と配線もコンピュータが勝手にやります。FPGA の内部では LUT は格子状に配置されているので、適切な位置と配線を自動的に選択するって事です。

実は、残りの作業も部分的に人間が行う事で最適化したりできますが、その辺はすでに職人芸の世界なのでなんとも。。基本的に僕はコンピュータ任せにすることが殆どです。

さて、デザイン入力ですが、普通のプログラミング言語でのプログラムみたいです。if とか then とか else もあります。が、しかし、非同期回路と同期回路の区別があるという重要な問題があります。

たとえば、同期回路は次のように書けます。

```
process(clk) if rising_edge(clk) then
      if c = '1' then
           B <= A;
           A <= B;
      end if; end if; end process;
```

みたいなソースコードがあったとすると、通常の感覚では、`c = 1` だったら `A = B` になりそうなもんですが、VHDL では、`<=` の左右は別世界だと考えます。つまり、この場合、A と B の内容が入れ換る、いいかえれば、右側がクロック前の世界、左側がクロック後の世界って事です。

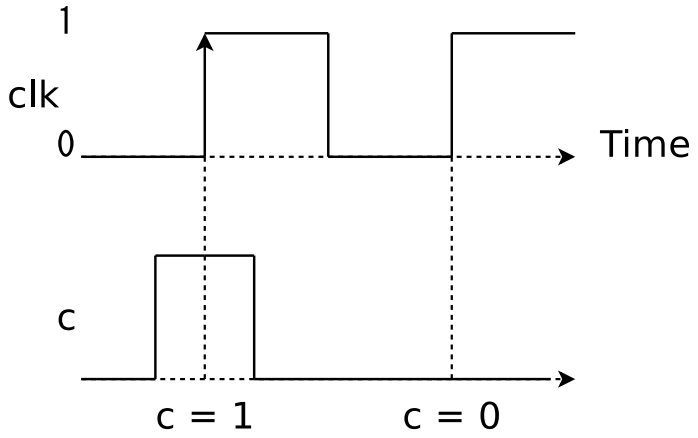


図 5　クロックとキャプチャ

例えば図 **5** のような波形でいうと、最初のエッジでは $c = 1$、次のエッジでは $c = 0$ として process の中身が処理される事になります。先に書いたように、論理回路の動作には遅延があるので、clk のタイミングバッチリに信号が変化するってことは (余程運が悪くないかぎり) ありません。

また、全ては並列に処理されます。process clk というのは、clk 信号に同期した同期回路って事をあらわしていますが、他の clk に同期した回路とも同時に動くことになります。

非同期回路は単に process の外に書きます。そうすれば「常に」実行される、要するに配線されるって事です。逆に process 文の中に書いたものは必要に応じてフリップフロップで信号を受けて、クロックに同期されます。

6　まとまらないまとめ

この先は。。

たった数頁の文章ですが、これだけの知識でコンピュータ作れます (本当に！)。が、実際には、『他人の都合に合わせる仕事』が努力の大半を占めます。たとえば、

- FPGA で使える機能ブロックを調べる
- USB とか TCP/IP みたいな各種プロトコルを実装する
- FPGA のタイミング制約や LUT の数の制約にあわせる

この辺の努力はかなり環境に依存するので、とにかくやりはじめてみて壁に突き当るしか突破する方法は無いように思います。

僕も、実際に USB ホストコントローラとか RAM インターフェースを作るにあたって壁に当ってきました。そういうのが知りたい人は是非 へ！

百萬石 2008年 春号 © 電気通信大学 MMA

2008 年 4 月 1 日　初版第 1 刷発行　【本書の無断転載を禁ず】

著　者　oku, moechar
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 2 階
http://www.mma.club.uec.ac.jp/ (IPv6)
http://delegate.uec.ac.jp:8081/club/mma/ (IPv4)
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室
表　紙　moechar

人人人人人人人人人人人人人人人人人人人人人＿
＞ゆっくりしていった結果がこれだよ！！！＜
￣^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y^Y￣

NetBSD

OpenBSD